Singletons and Services and Shutdowns Oh My

Passing data back and forth between activities on the Android platform is one of the more challenging aspects of understanding Android. Understanding how this data is managed by the OS is the difference between getting flooded with Error Reports and bad reviews and getting glowing praise and eternal gratitude. Well maybe not to that extent, but having a good background in passing data can make your code much easier to manage and cleaner to read (and for that you have my gratitude) So in passing the data there are essentially two types Persistent and strangely enough, Non-Persistent data types. The Persistent types are best handled by preferences, files, databases or content providers and they are pretty detailed in their operation. If you need to keep data over the length of more that one session consider using Persistent data stores. On the other hand, there are often times where transient data needs to be stored for one session and passed between activities. To handle that there are a couple of ways to handle it. This page details some of the methods, and it is repeated here for convenience;

How do I pass data between Activities/Services within a single application?

It depends on the type of data that you want to share:

Primitive Data Types

To share primitive data between Activities/Services in an application, use Intent.putExtras(). For passing primitive data that needs to persist use the Preferences storage mechanism.

Non-Persistent Objects

For sharing complex non-persistent user-defined objects for short duration, the following approaches are recommended:

The android.app.Application class

The android.app.Application is a base class for those who need to maintain global application state. It can be accessed via getApplication() from any Activity or Service. It has a couple of life-cycle methods and will be instantiated by Android automatically if your register it in AndroidManifest.xml.

A public static field/method

An alternate way to make data accessible across Activities/Services is to use *public static* fields and/or methods. You can access these static fields from any other class in your application. To share an object, the activity which creates your object sets a static field to point to this object and any other activity that wants to use this object just accesses this static field.

A HashMap of WeakReferences to Objects

You can also use a HashMap of WeakReferences to Objects with Long keys. When an activity wants to pass an object to another activity, it simply puts the object in the map and sends the key (which is a unique Long based on a counter or time stamp) to the recipient activity via intent extras. The recipient activity retrieves the object using this key.

A Singleton class

There are advantages to using a static Singleton, such as you can refer to them without casting getApplication() to an application-specific class, or going to the trouble of hanging

> an interface on all your Application subclasses so that your various modules can refer to that interface instead. **But, the life cycle of a static is not well under your control; so to abide by the life-cycle model, the application class should initiate and tear down these static objects in the onCreate() and onTerminate() methods of the Application Class** [my emphasis]

Being as I come from a java EE background the first decision was to use the singleton class and allow the instance to be available across the VM. The singleton design pattern is fairly well-known and is a good easy way to manage control across many activities. In some previous projects I have used them to manage HTTP connections, image caching and global application configuration to much success. However, this is the important part to consider when using singletons which is very important; **Android OS can and will terminate your singleton and not even tell you about it.** I highlighted that in bold because if your design depends on singleton patterns you naturally assume they are going to stay persistent through the VM. These are frustrating errors and difficult to track down and even more frustrating for your users. For instance consider this piece of code: In your main activity;

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
SuperSingletonManager.create(this);
}
```

And your singleton;

```
private static SuperSingletonManager instance;
private SuperSingletonManager(Context context)
{
// do stuff once
}
protected static SuperSingletonManager create(Context context) {
instance = new SuperSingletonManager(context);
}
protected static SuperSingletonManager getInstance() {
if (instance == null)
throw new NastyException("Oh God Why?");
return instance;
}
```

Seemingly you should be able to call *SuperSingletonManager.getInstance()* at any time and get access to the static instance. However, this isn't the case. If the launching activity is removed by the OS (it happens, a lot) while you are on another activity, that static instance will be gone. So when you make that *SuperSingletonManager getInstance()* call you are only going to get a nasty exception. This also means that if any of your functions in *SuperSingletonManager* make use of the *Context* those will throw errors. Ack. *But I really, really like singletons.* So do I. And far be it from me to tell you how to architect your code. The only stipulation is that the singleton should **abide by the lifecycle model.** We can do this by launching the singletons from a service and binding that service to the launching activity. To the Android purists, and common-sensists out there you might just say "Why not just use a service instead of a singleton?". Sure, makes sense but this post is about singletons and how to get them working and not common-sense. So here is the class that will do all that;

```
public class SingletonService extends Service {

private final ISingletonService.Stub mBinder = new ISingletonService.Stub() {

public void startSingletons() throws RemoteException {

initializeSingletons();

}

public void stopSingletons() throws RemoteException {

shutdownSingletons();

}

};
```

```
public IBinder onBind(Intent intent) {

return mBinder;

}

protected void initializeSingletons() {

SuperSingletonManager.initialize(getApplicationContext());

}

private void shutdownSingletons() {

SuperSingletonManager.shutdown();

}

}
```

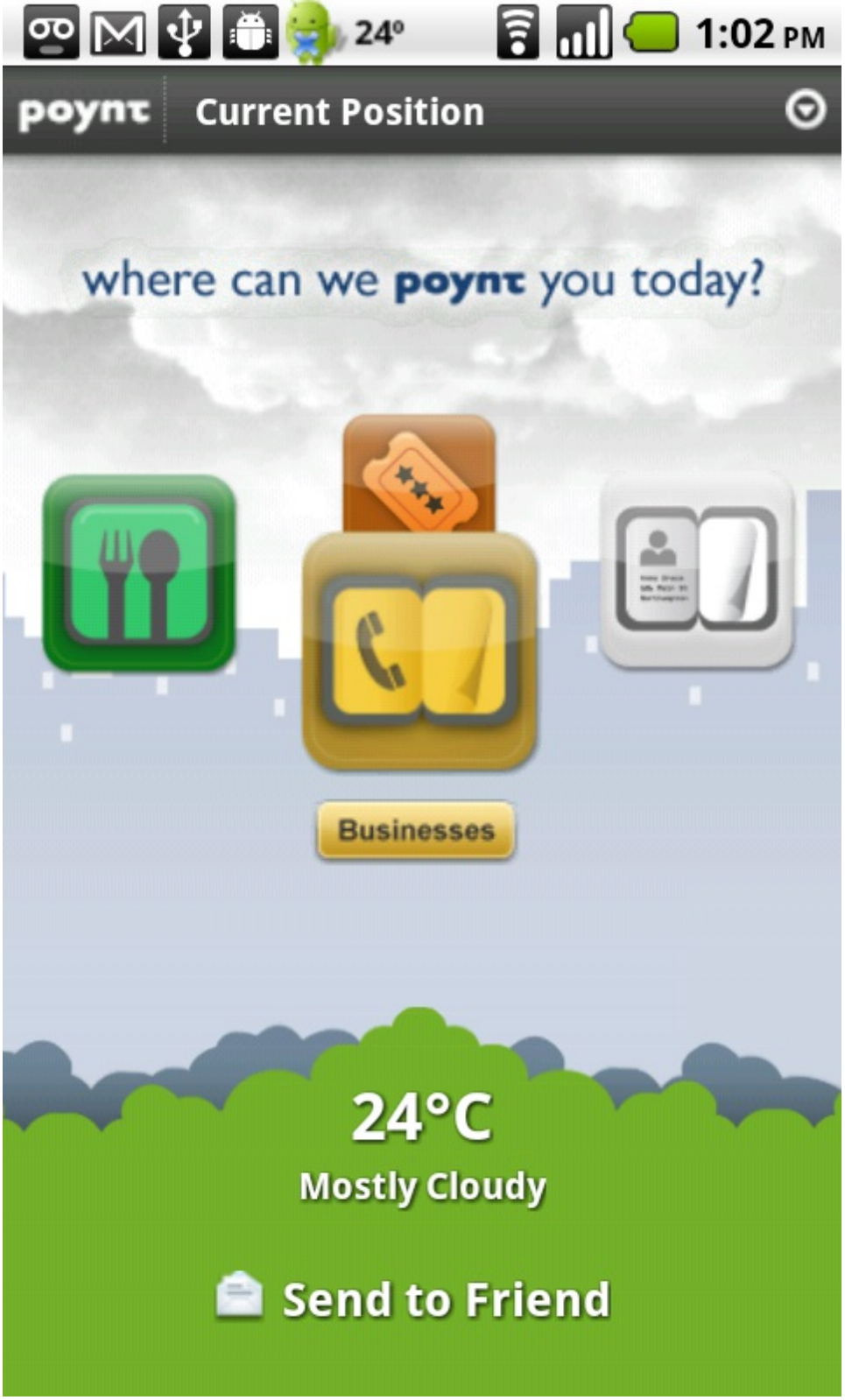
When you start your main activity bind the service using *bindService()* , and make a call to the *startSingletons()* method. This will launch your singletons under the lifecycle of the service. This will ensure they are active for the life of your application session. Also, make sure to *unbindService* when you are finished. Nobody likes developers that don't clean up after themselves. Good luck and if you actually use this method let me know! John

Posted via email from John Carpenter

Tagged as: Android, services, singleton No Comments

Poynt comes to Android

I've been helping the good people over at Multiplied Media with their Poynt for Android application for a couple of months now and I'm proud to see that they have launched the application.



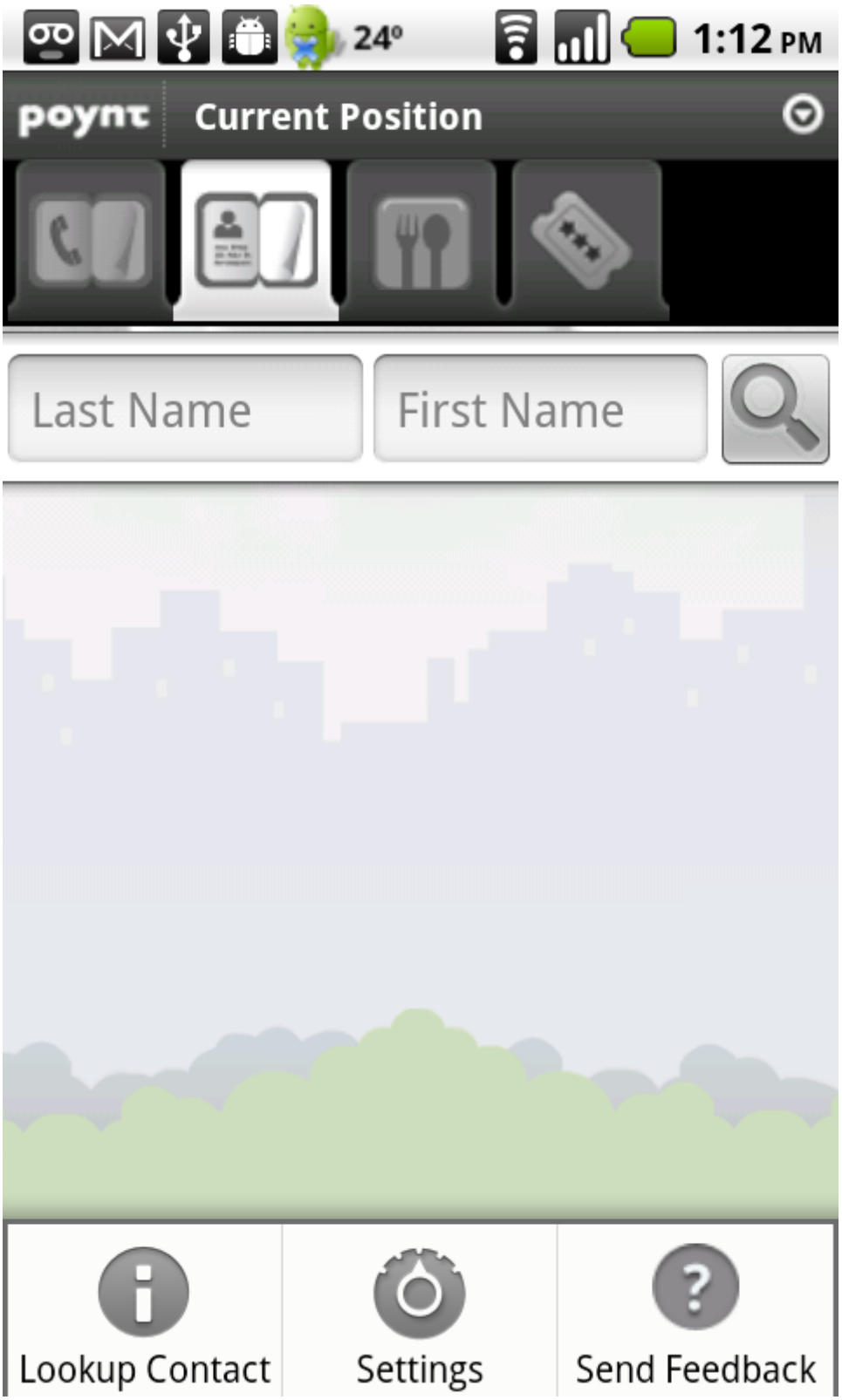
[Market Link]

The team did a great job putting the application together and hopefully it will be well received by the community. Although its still being distributed and talked about here are some great features that might not be immediately apparent.

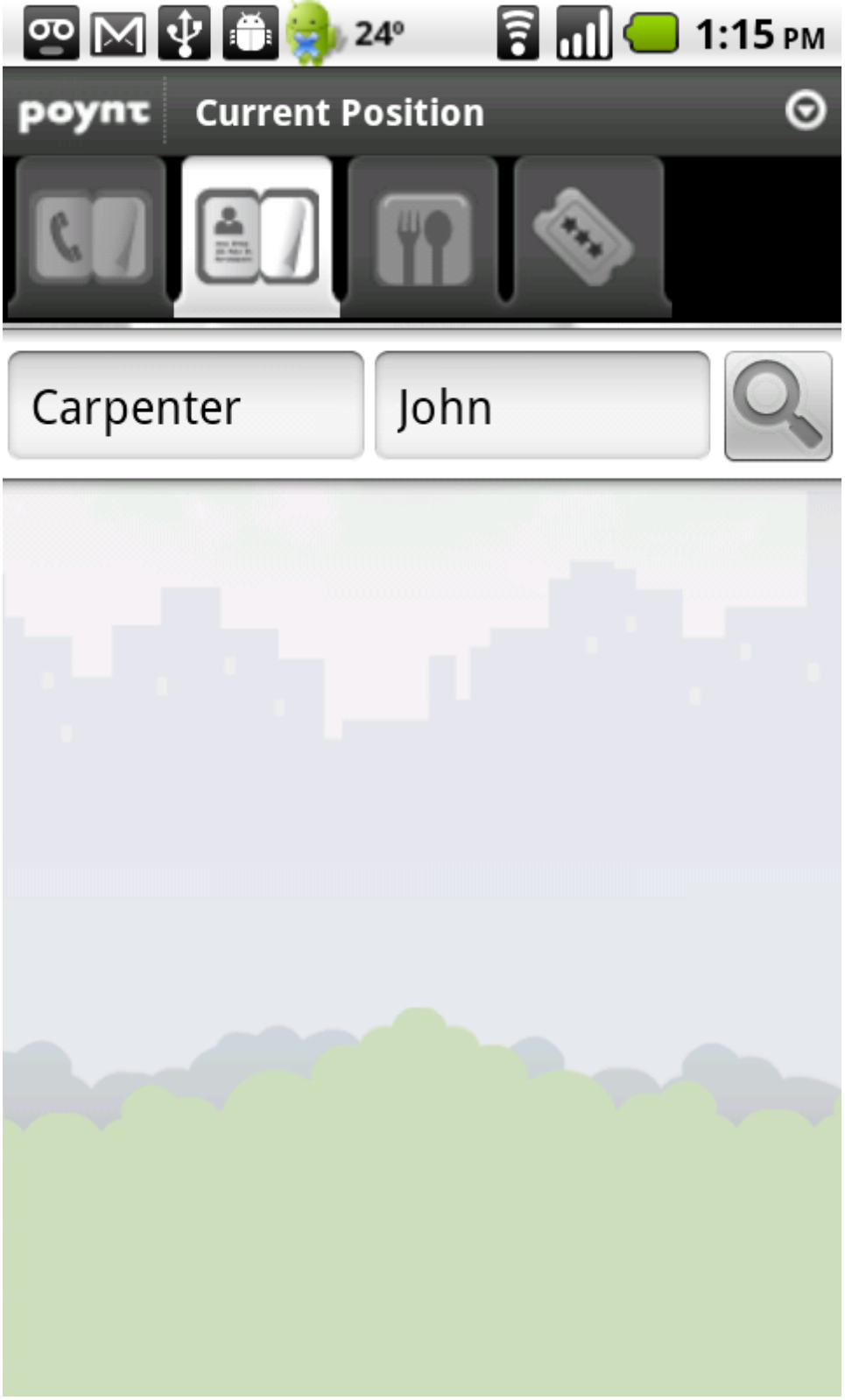
1. Look up a Contact in the White Pages Search

On 1.6+ devices, and in the countries where the White Page lookup is supported, Poynt has the ability to prepopulate the lookup fields from one of your existing contacts.

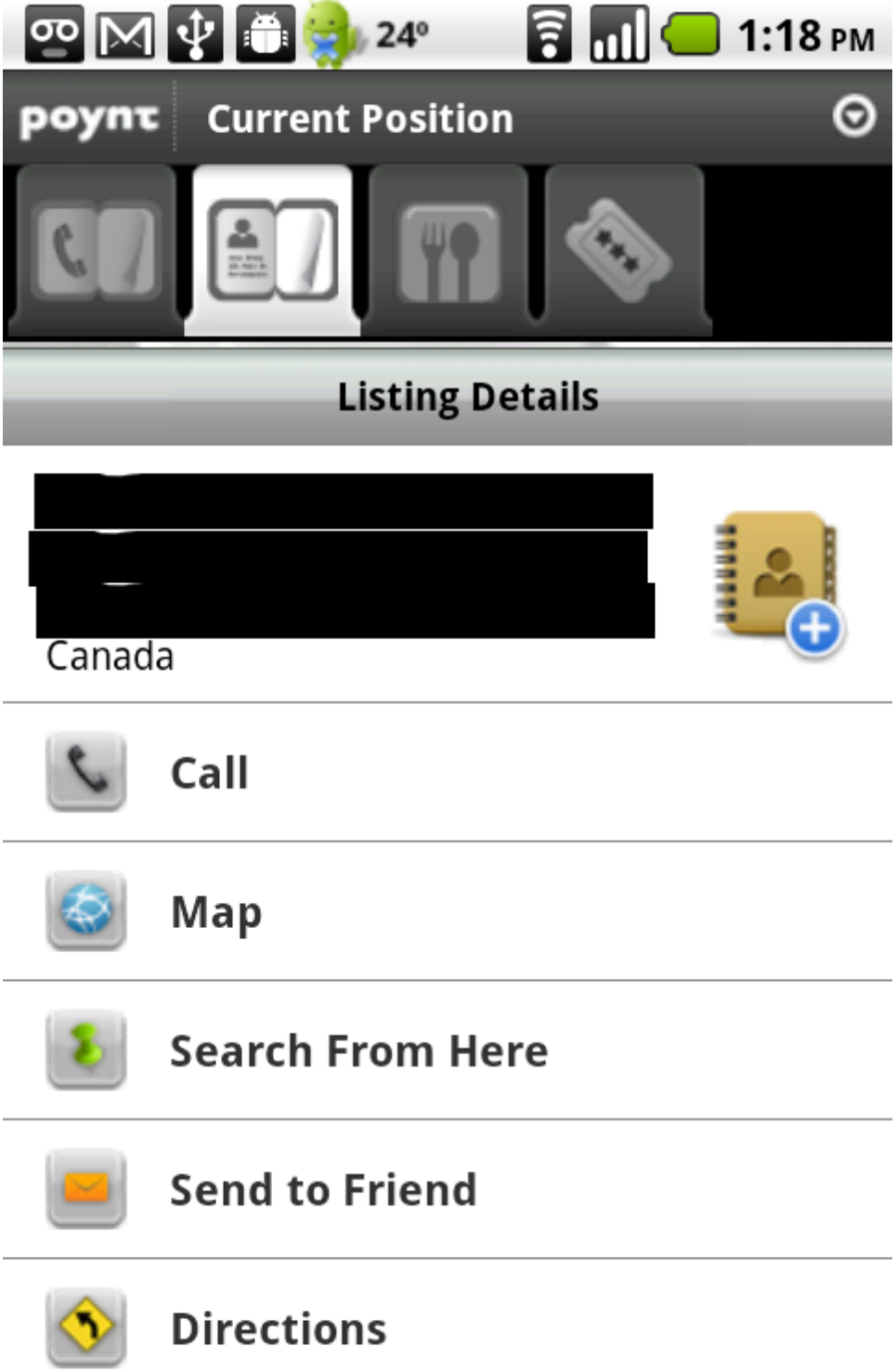
To get the function, select People from the Main Carousel, and select one of the lookup items. For this example, I will select "Lookup By Name".



Press the Menu key and there it is, "Lookup Contact". By pressing the menu item you will get your contact list to show up. Select one of the names and the menu will auto populate the fields.



Press the search icon and get the list of the nearby search results. Click on the appropriate search result and get the specific details.



Now, click the Add Contact icon on the right side, and Android will automatically merge the details from the White Pages lookup with your contact details.

So with a couple of clicks you can add the address, and phone number to your existing. contacts by using the People lookup in Poynt.

Look for more tips soon and make sure to download and try out Poynt for Android.

Filed under: Uncategorized No Comments

Starting Android Development — READ THIS FIRST

After releasing a couple of products into the Android marketplace you get a little more familiar with the OS and framework. Looking back you start to realize that Android has a number of methods and tools that, if you took the time to learn them first you would have saved yourself a ton of time.

Search after search related to problems I ran into always seemed to direct me to one page. The page didn't always have the answer to your problem but more than once it indicated that maybe I'm asking the wrong question in the first place!

Well here it is the ***single most important page in the entire Android framework***!

http://developer.android.com/guide/topics/fundamentals.html

I can't stress this enough. Read this page in its entirety. When you are finished go back to the start and read it again. Don't do any coding until you understand how your program gets executed!

Filed under: Uncategorized No Comments

Android Development and the Case of the Missing Google Maps Reference SDK

The Android Developer site has moved a bunch of links around to remove a lot of the Google branding from the community. Technically speaking Google drove the project and owns a lot of the development but it is an open-source community and probably should avoid proprietary and company specific codebases within its structure.

That at least what I assume they were thinking when they yanked the com.android.maps reference documentation from the site. For instance if you were looking for the MapController reference you would run a Google search for it and the first link is this:

http://code.google.com/android/reference/com/google/android/maps/MapController.ht

Which returns a bug fat 404. The proper link is at:

http://code.google.com/android/add-ons/google-apis/reference/index.html

Android/Google get it together! At least put a redirect page up on old links!

Hopefully that saves you some time with the SDK.

Filed under: Android, Location Based Services No Comments

Mistakes in Programming Projects

This should be required reading for any Software Project Manager.

http://www.stevemcconnell.com.nyud.net/rdenum.htm

Although for you jaded and older programming types you should play the "Been There" Bingo with that page. I've been on projects that scored well over 20/36!

Tagged as: programming

1 Comment

» seirtnE redlO

Author

My name is John Carpenter. I am a mobile enthusiast and developer out of Calgary, Canada. I (in)frequently write about programming problems I run into and hopefully that will be of value to you.

Recent

- Singletons and Services and Shutdowns Oh My
- Poynt comes to Android
- Starting Android Development — READ THIS FIRST
- Android Development and the Case of the Missing Google Maps Reference SDK
- Mistakes in Programming Projects

LinkedIn

If you want to see my LinkedIn profile, click on this button:

View my profile on LinkedIn

Follow me on Twitter

Sure fire way to make money in mobile, be a lawyer. http://bit.ly/bTvlg9 Who's suing who? 2010/10/06

Broken laptop video card. Bake at 385F for 7min. Remelt solder. Presto working laptop. Brilliant! 2010/10/05

Nokia, Google, BB, VZW and MSFT offering $13MM in developer contests. http://bit.ly/cR1lL5

http://bit.ly/cgoYkL Time to start coding! 2010/09/23

Tags

Add new tag Android Awards barcamp bugs contest ctia democamp first xkcd Gaming GDC GDC Mobile IGF iphone lbs lbs challenge logo management Mobile Monday navteq PhoneTag Elite presentation programming rogers services singleton Snocator SpotJots startups teams Trapster tutorial unlimited useful networks wikiwhere wireless

Go to top ↑